VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2016 Markéta Glatzová

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **Markéta Glatzová**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Railsformers s.r.o.
2. Struktura závěrečné zprávy:
a. Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b. Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c. Zvolený postup řešení zadaných úkolů
d. Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e. Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f. Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeňka Chmelíková, Ph.D.**

Konzultant bakalářské práce: Ing. Richard Lapiš

Datum zadání: 01.09.2015
Datum odevzdání: 29.04.2016

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 15. dubna 2016 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 15. dubna 2016

.....

Railsformers s.r.o.
IČ: 24704440 DIČ: CZ24704440
www.railsformers.com
info@railsformers.com

Ráda bych na tomto místě poděkovala celé firmě Railsformers, s.r.o. za spolupráci. Zvláště bych chtěla poděkovat jejímu jednateli Ing. Jiřímu Kubicovi za možnost vykonávat odbornou praxi právě v této firmě. Dále bych ráda poděkovala svému konzultantovi Ing. Richardu Lapišovi za vedení a ochotu, se kterou mi pomáhal při řešení problémů. Také bych chtěla poděkovat vedoucí mé bakalářské práce Ing. Zdeňce Chmelíkové, Ph.D. za odbornou pomoc a konzultaci při vytváření této práce.

Abstrakt

Cílem této bakalářské práce je popsat mé působení ve firmě Railsformers, s. r. o., se sídlem v Ostravě, ve které jsem vykonávala odbornou praxi. Práce obsahuje a popisuje zadané úkoly a jejich řešení, v některých případech také odůvodnění použití určitých řešení problémů. Na konci práce shrnuji praktické znalosti a dovednosti získané v průběhu studia, které jsem při praxi využila a také znalosti či dovednosti, které mi v průběhu odborné praxe scházely. Praxe se dělila na dvě části, v první části probíhalo zaučovaní a ve druhé části bylo mou náplní práce plnit projekty, které mi byly zadány. Drtivou většinu času jsem se věnovala části druhé.

Klíčová slova: Odborná praxe, Railsformers s. r. o., Ruby on Rails, Ruby, vývoj aplikací, testování, HTML, HAML, CSS, MySQL, SQLite 3, JavaScript

Abstract

The prurpose of this bachelor thesis is to describe my work in company Railsformers, s. r. o., with residence in Ostrava, where I administrated my practical training. The thesis contains and describes given tasks and their solution, in some cases also reasons for use of certain solutions. In the end I summarize practical knowledge and skills acquired during my studies, which I've utilized and also knowledge or skills I was short of. The practical training has consisted of two parts, during the first part the actual training took place, and during the second one my task was to fulfill given projects. The most of time I've been describing the second part.

Key Words: Professional Practice, Railsformers s. r. o., Ruby on Rails, Ruby, application development, testing, HTML, HAML, CSS, MySQL, SQLite 3, JavaScript

Obsah

Seznam použitých zkratek a symbolů

CSS	–	Cascading Style Sheets
MVC	–	Model View Controller
WWW	–	World Wide Web
SQL	–	Structured Query Language
SQLite3	–	databázový systém SQLite3
MySQL	–	databázový systém MySQL
MVC	–	model-view-controller
JS	–	Javascript
GUI	–	Graphical User Interface
AJAX	–	Asynchronous JavaScript and XML
SEO	–	Search Engine Optimization
QR	–	quick response
PHP	–	Hypertext Preprocessor
ASP	–	Association of Shareware Professionals
XML	–	eXtensible Markup Language
XHTML	–	eXtensible HyperText Markup Language
HTML	–	Hyper Text Markup Language
HAML	–	HTML Abstraction Markup Language

Seznam obrázků

Seznam výpisů zdrojového kódu

Úvod

Pro vykonání závěrečné práce jsem měla dvě možnosti. Rozhodovala jsem se mezi klasickou bakalářskou prací a bakalářskou prací formou praxe.

Po dlouhém zvažování všech pro a proti jsem se nakonec rozhodla pro absolvování individuální odborné praxe ve firmě Railsformers, s.r.o.

Rozhodla jsem se tak z mnoha důvodů, hlavním důvodem bylo to, že mi přišlo mnohem zajímavější vyzkoušet si své znalosti v praxi, prohloubit je a získat cenné zkušenosti, které jistě v budoucnu skvěle využiju, než se věnovat jen jedné určité problematice v pohodlí domova.

Ve firmě jsem pracovala na pozici Ruby on Rails programátor. V první části praxe jsem si četla příručky a návody převážně z internetu, ale i z odborné literatury a ve druhé části jsem plnila zadané projekty, které se týkaly vývoje webových aplikací v programovacím jazyce Ruby fungujícím na frameworku Rails.

V dalších kapitolách Vás podrobně seznámím s technologiemi2, které jsem při práci použila a s projekty3, které jsem během praxe řešila. Také popíšu postupy řešení a odůvodnění výběru některých technologií a způsobu řešení dané problematiky.

V neposlední řadě Vás seznámím s profilem společnosti 1 a celkově s frameworkem Ruby on Rails2.2.

1 O firmě Railsformers, s. r. o.

Obrázek 1: Logo společnosti

Počátkem roku 2012 byla Ing. Jiřím Kubicou založena firma Railsformers s.r.o., sídlící v Ostravě na adrese Technologická 372/2, 708 00 Ostrava - Pustkovec, která se zabývá převážně tvorbou internetových a intranetových systémů.

Vývoji webových aplikací a informačních systémů se věnují na profesionální úrovni již několik let. Specializují se na projekty větších rozsahů, jako jsou například různá enterprise řešení, sociální sítě, komunitní portály a jiné komplexní systémy, kde je naplno využito možností frameworku Ruby on Rails.

Nebojí se také použití AJAXu nebo různých efektových knihoven jako jQuery UI a nikdy nezapomínají na SEO. Již při vývoji pamatují na optimalizaci pro vyhledávače.

Většinu zakázek dělají pro zahraniční klientelu. Níže 1.1 Vám přiblížím jejich hlavní projekty.

1.1 Hlavní projekty

Firma má na svém kontě mnoho zajímavých projektů. Níže Vám přiblížím ty nejpodstatnější ze všech.

1.1.1 Hedurio

Hlavní produkt firmy. Koncovým uživatelům je poskytován na bázi cloudového řešení na bázi SaaS, čímž je zajištěna maximální dostupnost systému odkudkoliv a kdykoliv.

Mezi základní funkce systému řadíme například komplexní plánování stabilních objektů jakými jsou například obchodní řetězce, sklady, výrobní haly, atd., ale také otevřené akce, různé festivaly apod.

V rámci tohoto plánování je pomocí systému Hedurio zajištěn komplexní monitoring nejen budov, ale i strážných v terénu. Kontrolu obchůzek strážných zajišťuje QR kód, který strážní načítají při průchodu každého checkpointu pomocí tabletu, nebo chytrého telefonu.

Hedurio ale nezahrnuje jen monitoring objektů, ale nabízí také možnost vedení mzdových a účetních přehledů v rámci společnosti a zobrazení základního náhledu efektivního hospodaření podniku.

Dále Hedurio nabízí elektronický monitoring zásob ve skladu s aktuální databází počtu a stavu zboží. Podrobný přehled veškerých procesů na skladě včetně vstupů a výstupů společně s přehledem dodavatelů a odběratelů.

Hedurio je také aplikováno na nový ticketingový systém pro fotbalový klub Baník Ostrava. Vytváří databázi registrovaných fanoušku a eviduje je.

V budoucnu je plánováno propojení databáze fotek fanoušků s videem na stadionu sloužící k okamžité identifikaci osob při nevhodném chování. Veškeré procesy jsou také poskytovány a zobrazeny online.

1.1.2 sMoneybox

Projekt sloužící k domácímu online vedení účetníctví. Pomáhá uživatelům mít přehled nad svými příjmy a výdaji kdykoliv a odkudkoliv. Registrace je anonymní a zcela zdarma.

1.1.3 sÚčto

Súčto je projekt sloužící k jednoduchému a přehlednému vedení fakturací ale také účetnictví. V základním tarifu je služba nabízena zcela zdarma. Jednoduše vytváříte faktury a evidujete Vaše výdaje online a máte tak neustálý přehled o Vašem podnikání a šetříte tak svůj čas i peníze.

1.1.4 sRecepty

Stránky zabývající se vším, co souvisí s vařením, recepty a kořením. Stránky jsou převážně tvořeny uživateli, kteří zde vkládají své recepty, ale obsahují také mnoho zajímavých článků souvisejících se stravováním a jídlem obecně.

Na stránkách se občas pořádají i různé soutěže, v minulosti zde bylo možno například vyhrát lístky na festival jídla FOODPARADE.

1.2 Projektově.CZ

Projekt projektově není projekt společnosti Railsformers s.r.o. ale je to projekt společnosti Projektově.CZ s.r.o.

Do své práce ho však zahrnuji, jelikož jsem při vykonávání praxe tuto WWW stránku velice často využívala. Na Projektově jsem dostávala veškeré úkoly.

Celé to slouží ke zpřehlednění zadaných úkolů a také se zde zapisuje kolik času jsme nad daným úkolem strávili a co jsme vše zvládli za tuto dobu udělat. U úkolu lze také nastavit kdo může daný úkol sledovat.

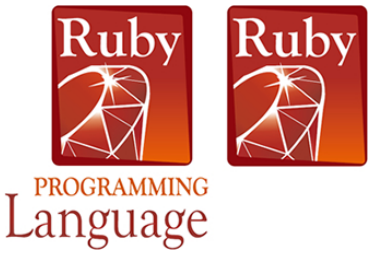
2 Použité technologie

V této kapitole bych Vám ráda představila pár technologií, se kterýma jsem se při vykonávání odborné praxe setkala.

Začala bych úvodem, co to vlastně je jazyk Ruby a framework Ruby on Rails, pokračovat budu popisem dalších technologií, jako jsou například JS, CSS, používání gitu a podobně.

Také v této kapitole porovnám pár podobných technologií a odůvodním, proč jsem si vybrala danou technologii.

2.1 Ruby



Obrázek 2: Ruby

Tvůrcem Ruby je pouze jediný člověk a tím je Jukihiro Macumoto. Ruby je jednoduchý interpretovaný skriptovací programovací jazyk, který nemá přílíš složitou syntaxi, ale přesto je to velmi výkonný objektový programovací jazyk.

V Ruby je vše objekt. Nejčastěji je využíván na platformě Linux, ale je znám i díky své přenositelnosti mezi různými platformami.

Poznámka 1 Jednoduchý kousek kódu, který umožní vytvoření nového profilu a při úspěchu vypíše upozornění o úspěšném vytvoření

```
def create
  @profile = Profile.new(profile_params)
  respond_to do |format|
    if @profile.save
      format.html { redirect_to @profile, notice: 'Profile was successfully
          created.' }
    else
      format.html { render :new }
    end
  end
end
```

Výpis 1: Ukázka Ruby kódu

2.2 Ruby on Rails

Autorem je dánský programátor David Heinemeier Hansson. Ruby On Rails je framework pro vývoj webových aplikací v jazyce Ruby využívající architekturu MVC.

Základní myšlenkou architektury MVC je oddělení logiky od výstupu. Zjednodušeně řečeno MVC dělí aplikaci na 3 logické části tak, aby šlo jednotlivé části samostatně upravovat s co nejmenším dopadem na části ostatní.

Těmito částmi jsou:

- Model
- View
- Controller

Nyní Vám blíže vysvětlím princip jednotlivých částí

2.2.1 Model

Modely obsahují většinovou funkčnost aplikace(aplikační logika). Reprezentují nám data ale také i pravidla práce s těmito daty.

V Rails slouží modely především k interakcí s danou tabulkou v databázi a pro ukládání pravidel této interakce. Většinou odpovídá jedna tabulka v databázi jednomu modelu v aplikaci.

2.2.2 View

Views nebo také pohledy nám upravují výsledné uživatelské rozhraní naší aplikace. Mohou to být jak prvky HTML s částmi Ruby kódu tak také novější HAML. Níže v mé práci Vám přinesu porovnání HAML a HTML2.5.

2.2.3 Controller

Controllery fungují zjednodušeně řečeno jako „lepidlo“ mezi modely a views.

V Rails controllery vlastně získávají data z modelů a odesílají je do views, kde jsou zobrazovány. Můžeme tedy v controllerech upravovat to, jaké data budou ve views zobrazeny. To znamená, že můžeme napsat například v controlleru různé podmínky, které když budou splněny, zobrazí se různá data.

2.3 Ruby Gems

Ruby Gems je vlastně balíčkovací systém pro Ruby, umožňující instalaci knihoven. Jelikož je to v Ruby, tak je to také multiplatformní, to znamená, že je snadno přenositelný mezi různými platformami. Všechny gemy vkládáme do souboru Gemfile.

Gem může vypadat například takto:

```
gem 'bootstrap-sass', '~> 3.3.6'
```

Výpis 2: Gem

Na příkladu, můžeme vidět, že si můžeme nainstalovat i přímo námi vybranou verzi gemu. Následně už jen stačí v terminálu zadat příkaz bundle install a můžeme s gemy dále pracovat a zakomponovat je do svého projektu.

Ráda bych dále vypsala pár gemů, které jsem používala nejčastěji.

2.3.1 gem 'devise'

Tento gem slouží pro registraci a přihlašování, ale lze také pomocí něho jednoduše provádět změnu hesla, potvrzení registrace přes email, nastavovat minimální a maximální počet znaků hesla a další věci.

2.3.2 gem 'simple_form'

Slouží pro úpravu formulářů, pomocí tohoto gemu můžeme docílit pěknějších formulářů.

2.3.3 gem 'kaminari'

Tento gem se používá pro stránkování. Stránkování pak lze i pokročile upravovat ve views.

2.3.4 gem 'carrierwav'

Pomocí tohoto gemu můžeme uploadovat soubory, používala jsem ho pro vkládání obrázků na stránky.

2.3.5 gem 'searchkick'

Searchkick nám pomáhá při vyhledávání. Řeší také pokročilejší problémy, které mohou při vyhledávání nastat.

Searchkick funkce:

- vyplývající slova - tomatoes odpovídá i tomato
- speciální znaky - jalapeno odpovídá jalapeño
- bílé znaky(mezery) - dishwasher odpovídá dish washer
- pravopisné chyby - zuchini odpovídá zucchini
- vlastní synonyma - qtip odpovídá cotton swab

2.4 Bootstrap vs Foundation

Jak Bootstrap tak i Foundation slouží pro úpravu vzhledu výsledné aplikace. Pro práci s Bootstrapem i Foundationem musíme prvně vložit příslušný gem do Gemfile a nainstalovat.

Poté se můžeme podívat na dokumentaci a zjistit, co přesně nám tyto gemy nabízí a jak se s němi pracuje.

Při své praxi jsem měla možnost vyzkoušet obě možnosti. Nejnovější verze Bootstrapu je 4.0.0-alpha a u Foundationu je to verze 6.

Jejich hlavním společným znakem je to, že pracují s mřížkou. Oba mají 12 sloupců, syntaxe se ale liší.

Syntaxe pro jeden sloupec:

```
<!-- Bootstrap 4.0.0-alpha -->

.col-xs-1
.col-sm-1
.col-md-1
.col-lg-1
.col-xl-1

<!-- Foundation 6 -->

.small-1.columns
.medium-1.columns
.large-1.columns
.[custom]-1.columns
.one.column
```

Výpis 3: Jeden sloupec porovnání syntaxe Bootstrap vs Foundation

Syntaxe pro dva sloupce:

```
<!-- Bootstrap 4.0.0-alpha -->

.col-xs-2
.col-sm-2
.col-md-2
.col-lg-2
.col-xl-2

<!-- Foundation 6 -->

.small-2.columns
.medium-2.columns
.large-2.columns
.[custom]-2.columns
.two.columns
```

Výpis 4: Dva sloupce porovnání syntaxe Bootstrap vs Foundation

Porovnání syntaxe kontejnéru:

```
<!-- Bootstrap 4.0.0-alpha -->

<div class="container">
nebo
<div class="container-fluid">

<!-- Foundation 6 -->

<div class="row">
<div class="container">
```

Výpis 5: Kontejnéry porovnání syntaxe Bootstrap vs Foundation

Syntaxe řádků:

```
<!-- Bootstrap 4.0.0-alpha -->

<div class="row">
<div class="col-xs-2 col-sm-4 col-md-6 col-lg-7 col-xl-8">...</div>
<div class="col-xs-10 col-sm-8 col-md-6 col-lg-5 col-xl-4">...</div>
</div>

<!-- Foundation 6 -->

<div class="row">
<div class="small-8 medium-6 large-4 [custom]-2 columns">...</div>
<div class="small-4 medium-6 large-8 [custom]-10 columns">...</div>
</div>
```

Výpis 6: Porovnání syntaxe řádků Bootstrap vs Foundation

Lépe se mi však pracovalo s Bootstrapem a myslím, že je výhodnější používat Bootrstrap hlavně z důvodu dostupnějších již hotových šablon(Templates).

2.5 HTML vs HAML

Nejprve, musíme vložit gem haml do Gemfile a nainstalovat. HAML je náhrada za ERB. To znamená, že jakýkoliv soubor ve složce app/views lze přepnout na HAML pouhou změnou přípony souboru.

Například:

- app/views/home/index.html.erb

změníme na:

- app/views /home/index.html.haml

Nyní můžeme přepsat syntaxi.

Kód napsaný v HAML zabíra méně místa, než kód napsaný v HTML.

Ukázky pro porovnání:

```
<!-- 1. Priklad -->
<!-- HTML -->
<strong class="code" id="message">Ahoj!</strong>

<!-- HAML -->
%strong{:class => "code", :id => "message"} Ahoj!

<!-- 2. Priklad -->
<!-- HTML -->
<div class='content'>Ahoj!</div>

<!-- HAML -->
.content Ahoj!

<!-- 3. Priklad -->
<!-- ERB -->
<div id='content'>
<div class='left column'>
<h2>Vitejte na strankach!</h2>
<p><%= print_information %></p>
</div>

<div class="right column">
<%= render :partial => "sidebar" %>
</div>
</div>

<!-- HAML -->
#content
.left.column
%h2 Vitejte na strankach!
%p= print_information

.right.column
= render :partial => "sidebar"
```

Výpis 7: Porovnání HTML vs HAML

U HAMLU je důležité zarovnání, stačí o jednu mezeru méně, nebo navíc a už to nebude fungovat. Pro někoho to může být i výhoda a může mu to připadat přehlednější, ale já raději pracovala s HTML jelikož sem neměla moc času vžít se do HAMLU a měla sem často problémy s tím, že jsem netušila, kde jsem to v kódu špatně zarovnala.

2.6 Javascript

JavaScript® je lehký, multiplatformní, objektově orientovaný skriptovací jazyk a je nejlépe známý jako interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky.

Obvykle nám pomáhá ovládat různé interaktivní prvky GUI, nebo nám pomáhá při výtváření různých animací a efektů obrázků.

Základní syntaxe je záměrně podobná jak Javě tak také C ++ z důvodu usnadnění, abychom se nemuseli učit nový jazyk. Jazykové konstrukce, jako je if, for a while cykly, switch a try-catch bloky fungují stejně jako v těchto jazycích (nebo velice obdobně).

JS může fungovat jako procedurální a objektově orientovaný jazyk. Program v JS se obvykle spouští až po stažení WWW stránky z Internetu (na straně klienta), na rozdíl od ostatních jiných interpretovaných programovacích jazyků jako jsou například PHP a ASP, které se spouštějí na straně serveru ještě před stažením z Internetu.

Z tohoto důvodu jsou zde jistá bezpečností omezení, JS například nemůže pracovat se soubory, aby tím neohrozil soukromí uživatele. Vé svých projektech jsem JS využila třeba pro zobrazování hvězdiček v hodnocení, nebo pro zobrazení mapy na kontaktní stránce.

2.7 CSS

CSS je zkratka pro kaskádové styly, pomocí kterých můžeme napsat popis způsobu zobrazení daných elementů na stránkách napsaných v jazycích HTML, XHTML, XML a HAML. CSS nám umožňuje úsporu času, jelikož můžeme vytvořené třídy využívat kdykoliv je potřebujeme a nemusíme parametry nastavovat pořád dokola. Extérní styly jsou uloženy v souborech CSS.

Ukázka syntaxe:

```
.social-icon{
display: block;
width: 60px;
height: 60px;
}
```

Výpis 8: Ukázka syntaxe CSS

Tato třída social-icon definuje uspořádání prvků v této třídě do bloku a upravuje šířku a výšku prvků na 60px. Můžeme si všimnou, že každý blok deklarací obsahuje deklarace oddělené

středníky ; a každá deklarace sestává z identifikátoru vlastností, následuje dvojtečka : a hodnota dané vlastnosti. Může také následovat označení !important, které zvyšuje sílu deklarace.

Toto označení použijeme nepříklad pokud na prvek používáme i jinou třídu, která deklaruje stejný identifikátor vlastností.

2.8 MySQL vs SQLite3

Jsou to databázové systémy. Musíme mít nainstalovaný v projektu daný gem. Jejich funkce nejsou vůbec stejné. SQLite je vložená databáze, která nemá žádné síťové funkce (dokud je nepřidáme).

Pokud tedy potřebujeme:

- přístup k síti - například přistupujeme z jiného počítače
- jakýkoliv stupeň souběžnosti - například, pokud si myslíme, že je pravděpodobné, že budeme spouštět několik dotazů najednou, nebo spustíme pracovní zátěž, která má spoustu vyběrů a několik aktualizací a chceme aby to probíhalo hladce
- velké využití paměti, například do vyrovnávací paměti části naší 1TB databáze v naši 32GB paměti.

Musíme použít MySQL. Na praxi jsem používala hlavně MySQL proto, že byla potřeba databáze, která funguje na serveru. SQLite je ale velmi pěkný kus softwaru. Je to malá knihovna, která rozjíždí SQL na lokálních souborech.

2.9 Git

Git vytvořil Linus Tovalds, původně pro vývoj jádra Linuxu, ale časem se vyvynul až do samostatně použitelného systému správy verzí.

Je to tedy systém na ukládání změn v projektu, který nejlépe poslouží programátorům pracujícím v týmu na stejném projektu(modulární programování). Sdílí pak spolu jeden projekt a současně na něm pracují. Toto je možné díky nejdůležitější funkci větvení. Pomocí větvení je možné vytvoření více větví, které jsou na sobě navzájem nezávislé.

Díky této nezávislosti si může programátor vytvořit svou větev, ve které bude řešit svou část projektu a s hlavní větví projektu ji sloučí až bude větev dostatečně optimalizována.

Git si nainstalujeme a následně musíme git také nastavit. Provádí se to pomocí git config. Nastavit bychom měli hlavně svou totožnost.

```
git config --global user.name "Marketa Glatzova"
git config --global user.email marketa.glatzova@priklad.com
```

Výpis 9: Git nastavení totožnosti

Další základní věc o které bych chtěla mluvit je získání repozitáře a to lze hned dvěma způsoby.

- Vezmeme existující projekt nebo adresář a následně ho importujeme do systému Git.

```
echo "# nazevprojektu" >> README.md
git init
git add README.md
git commit -m "prvni komentar
git remote add origin https://github.com/JmenoNaGitu/nazevprojektu.git
git push -u origin master
```

Výpis 10: Git existující projekt

- Naklonujeme již existují Git repozitář z jiného serveru

```
git clone git://github.com/JmenoNaGitu/nazevprojektu.git
```

Výpis 11: Git klónování projektu

3 Projekty a úkoly řešené v průběhu praxe

Níže bych ráda popsala průběh své praxe a jednotlivé projekty a úkoly, které jsem měla na starosti.

3.1 Zaučování a tvorba blogu

První část praxe jsem se zaučovala a zkoušela si nabité znalosti při tvorbě prvního blogu.

Tato část ale dlouho netrvala. Dostala jsem týden na pročítání guide[8]. Zaučování probíhalo ale také mimo mou pracovní dobu v mém volném čase. Vypůjčila jsem si knihy z knihovny. Procházela jsem dvě knihy o Ruby on Rails[14][15] a jednu o práci s linuxem[13], jelikož jsem ho předtím na svém notebooku nikdy neměla a mé znalosti s tímto systémem byly pouze minimální.

Po dočtení guide jsem si měla zkoušet nabité vědomosti při tvorbě blogu podle návodu. [11] Na tomto blogu jsem si zkoušela pracovat s gemy. Pomáhaly mi při tom videa na Railscasts[9]. Vyzkoušela jsem si například simple_form pro lepší formuláře, devise pro přihlašování, foundation pro úpravu vzhledu a carrierwave pro nahrávání souborů.

Dále jsem si pročítala návody na práci s Gitem, založila jsem si učet na gitlabu a githubu, nainstalovala Git, vložila na stránky vygenerovaný SSH klíč, provedla nějaké nastavení Gitu, stáhla si editor emacs a zkusila podle příručky ze stránek práci s projektem.

Nakonec jsem dostala knihu v pdf o psaní testů pomocí Rspec[12], tu jsem si prošla a zkoušela testy na vytvořeném blogu. Tímto hlavní část zaučování skončila a dále jsem se učila rovnou už při pomoci s projektem a především poté vytvářením projektů zadaných mně samostatně.

3.2 Pomoc s dokončením projektu inzerce

V této části praxe jsem se připojila k ostatním abych jim pomohla s dokončováním jejich projektu inzerce.

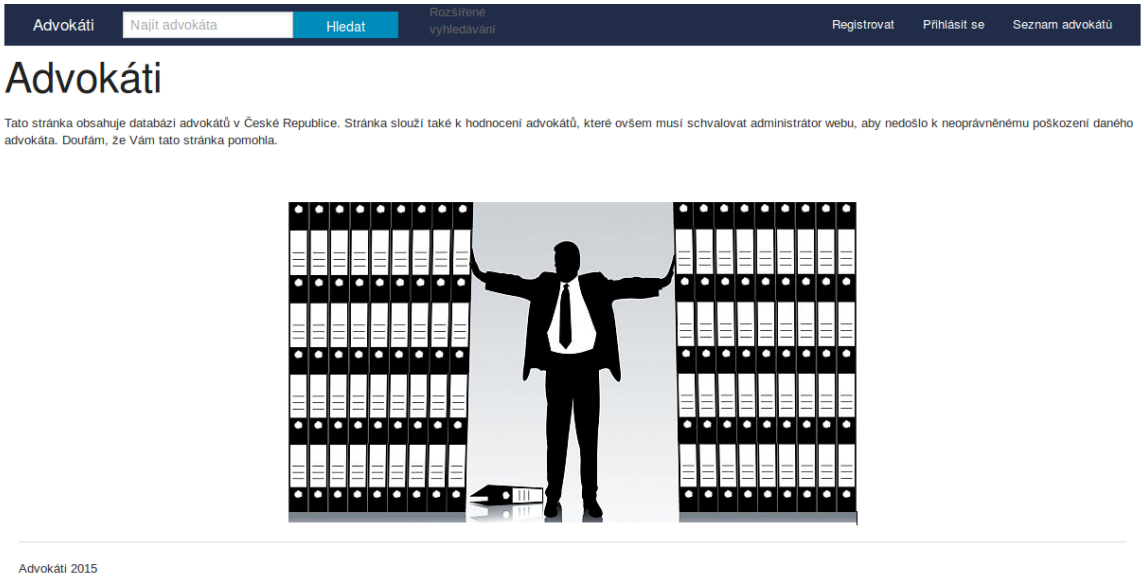
Jelikož jsem měla už pročtenou knihu o testech a něco i odzkoušené, tak jsem jim pomohla s psaním pár testů.

Poté jsem ještě pomohla dodělat překlady a zkoušela jsem zprovoznit přihlašování přes facebook, ale to mi tenkrát ještě nešlo a dostala jsem už svůj projekt, tak jsem se pustila do práce na něm.

3.3 První samostatný projekt hodnocení advokátů

Hodnocení advokátů byl první projekt, na kterém jsem pracovala již samostatně. Zadání projektu jsem dostala od svého konzultanta. Projekt má sloužit k přidávání hodnocení advokátů. Je to v podstatě něco na způsob znamylekar.cz.

Po příchodu na stránky uživatel uvidí hlavní stranu s informacemi k čemu vlastně webová stránka slouží a nahoře v navigaci má odkaz na registraci.



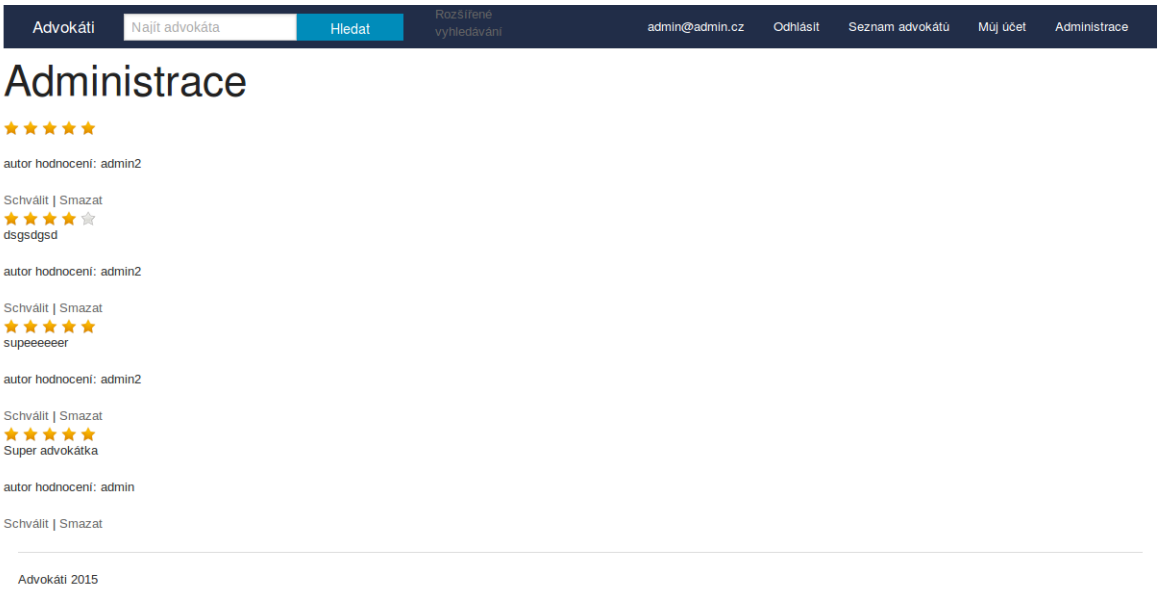
Obrázek 3: Advokáti, úvodní strana

Nepřihlášený uživatel může také procházet již přidané advokáty a pročítat si jejich hodnocení. Bez registrace ale nelze přidávat nové hodnocení.

Registrace probíhá velice jednoduše. Uživatel zadá svou přezdívku, emailovou adresu, heslo a heslo znova pro kontrolu. Fungují zde také funkce jako například zapomenuté heslo, možnost potvrzovacího odkazu, který přijde na email a samozřejme také změna hesla, která je možná kdykoliv a jednoduše po přihlášení na stránky kliknutím na emailovou adresu nahoře v navigaci.

Změnit se na této stránce dá také emailová adresa zadána při registraci. Přihlašování na stránky pak už po registraci probíhá pomocí emailové adresy a hesla. Vše je zajištěno pomocí gemu devise. Minimální a maximální počet znaků hesla lze zcela jednoduše v kódu upravit.

Po registraci tedy může uživatel přidávat nové hodnocení, ale u advokáta se zobrazí až poté, co ho schválí administrátor webu. Administrátora si vytváříme přímo v konzoli a dále se již může přihlašovat pomocí fromuláře jako běžný uživatel. Pouze administrátor může přidávat nové advokáty. Administrátor vidí veškeré nové celkové hodnocení a pokud hodnocení schválí tak se u odvokáta následně veřejně zobrazí. Pokud je ale hodnocení například příliš urážlivé, nebo jinak nevhodné tak ho administrátor rovnou smaže a hodnocení se u advokáta nikdy nezobrazí.



Obrázek 4: Advokáti, administrace

Advokáti se mohou hodnotit hned dvěma způsoby. Jeden způsob je advokáta ohodnotit celkově. V praxi to vypadá, tak, že mu udělíme určitý počet hvězdiček a napíšeme své zkušenosti také formou textu, toto hodnocení se pak započítává do celkového hodnocení, které se průběžně s přibývajícími hodnoceními aktualizuje. Vidíme také celkový počet těchto hodnocení. U celkového hodnocení je také vždy zobrazena přezdívka autora hodnocení. Pokud není u daného advokáta ještě žádné hodnocení zobrazí se nám hláška: "zatím zde nejsou žádné hodnocení, chceš být první?"a hned pod touto hláškou následuje odkaz k přidání nového hodnocení.

Po kliknutí na přidání celkového hodnocení se nám zoobrazí nová stránka a po uložení hodnocení nás to přesměruje zase zpátky na právě ohodnoceného advokáta. Hodnocení je správně přiřazováno k advokátům pomocí id, které se ukládá v databázi. Funkčnost tohoto propojení jsem si také kontrolovala přímo v konzoli. Aby se nám zobrazovaly hvězdičky a nikoliv pouze čísla od jedné do pěti jsem docílila napsáním scriptu a JQuery Raty[10].

Stáhla jsem soubor z githubu a soubor jquery.raty.js vložila do složky javascripts v mé aplikaci. Dále jsem do složky images vložila obrázky star-half.png, star-on.png, star-off.png. Následně stačilo už jen upravit view.

Do view jsem přidala tento script:

```
<script>
$('.star-rating').raty({
 path: '/assets/',
 readOnly: true,
 score: function() {
 return $(this).attr('data-score');
 }
});
</script>
```

Výpis 12: Script na zobrazení hvězdiček

Script má přístup k assets, je pouze pro čtení a vrací nám dané hodnocení(data-score).

K úplné funkčnosti již stačilo výpis "obalit"do tohoto divu.

```
<div class="star-rating" data-score= <%= review.rating %> ></div>
```

Výpis 13: Třída star-rating

Script jsem také musela vložit do view, ve kterém se hodnocení píše. Zde vypadá script lehce odlišně:

```
<script>
$('#star-rating').raty({
 path: '/assets/',
 scoreName: 'review[rating]'
});
</script>
```

Výpis 14: Script na zobrazení hvězdiček s mozností změny

Hlavní rozdíl je v absenci readOnly: true, tato část zde být nemůže, jelikož před uložením musíme mít možnost počet hvězdiček měnit.

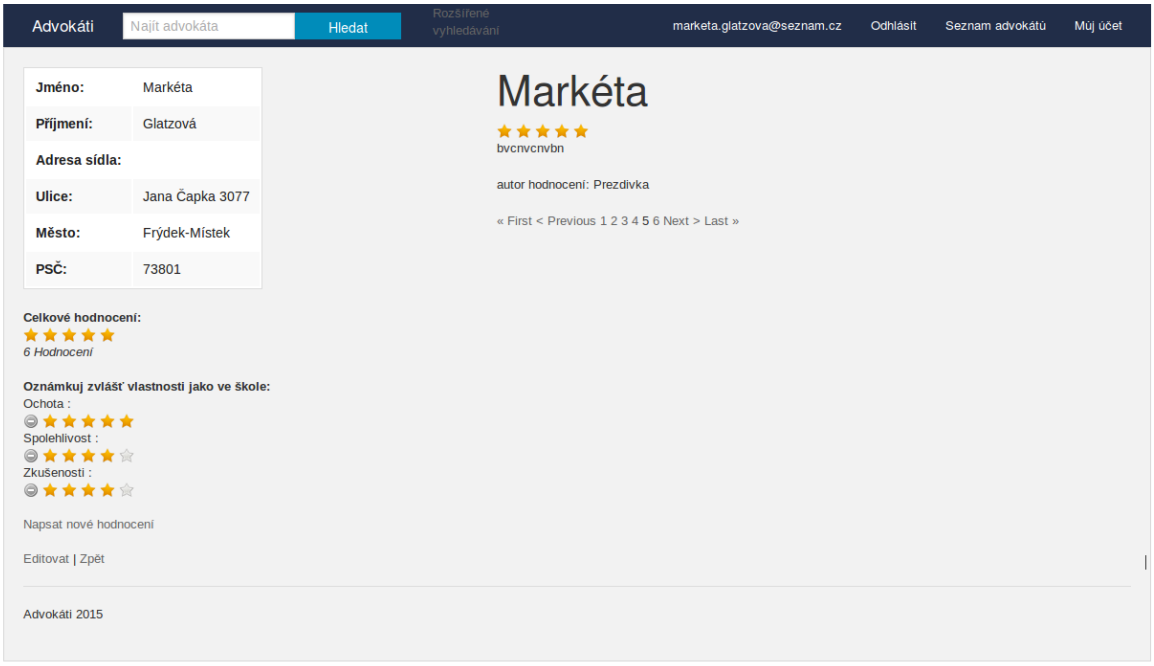
Dále jsem ještě musela přepsat část kódu, ve které se zobrazovaly číslice následující části kódu:

```
<div class="field">
<div id="star-rating"></div>
</div>
```

Výpis 15: Nahrazení číslic hvězdičkami ve view pro nové hodnocení

Druhým způsobem je zvlášť a zcela nezávisle ohodnotit jeho schopnosti, hodnotí se ochota, spolehlivost a zkušenosti.

Stránkování hodnocení je řešeno pomocí gemu kaminari. Uživatel má také zvlášť stránku s výpisem všech jeho dosavadně udělených hodnocení, které jsou rozděleny do dvou částí a to do čekájících na schválení a schválených hodnocení. Může tedy jednoduše kontrolovat zda je jeho hodnocení již schválené, nebo nikoliv.



Obrázek 5: Advokáti, view

Dále bych ráda popsala vyhledávání. Vyhledávání je umístěno přímo v navigaci nahoře a máme zase dva způsoby jak advokáty vyhledat. Můžeme buď zadat pouze klíčové slovo a kliknout na "Hledat", nebo pomocí odkazu na rozšířené vyhledávání se dostaneme na další stránku, na které můžeme zadat do formuláře pro vyhledávání jednotlivé parametry vyhledávání. Těmito parametry může být jakýkoliv údaj uložený v databázi při vytvoření daného advokáta. Každý advokát je v databázi veden pod svým jménem, příjmením a úplnou adresou sídla, která se skládá z města, ulice, čísla ulice a poštovního směrovacího čísla. Znamená to tedy, že můžeme jednotlivé advokáty vyhledat i pomocí určitého poštovního směrovcího čísla, ulice atd. Po zadání parametrů stačí jen kliknout na tlačítko potvrdit a pod formulářem se nám již zobrazí veškéré výsledky vyhledávání. Při tvorbě vyhledávání jsem využila gem searchkick. Vzhled stránek je zde řešen pomocí gemu Foundation. Design je jednoduchý, ale přehledný.

3.4 Druhý samostatný projekt registr nájmů

Projekt registr nájmů byl můj nejrozsáhlejší projekt, který mi zabral nejvíce času z mé praxe. Projekt mi byl zadán přímo panem Ing. Kubicou.

Dle NOZ (Nového občanského zákoníku) nemůže nájemce dále věci pronajímat, což ale nemusí třetí strana vědět. Proto jsem dostala za úkol udělat projekt, kde půjde tyto věci najít.

Primárně je projekt určen pro ochranu/kontrolu toho, kdo si věc pronajímá, vlastník moc chraněný není, každopádně tím, že to zveřejní a lidé možná časem budou tento nástroj používat, tak nemá garanci, ale šanci, že si jeho věc někdo ověří. Projekt obsahuje tyto funkce, které budu níže trochu podrobněji popisovat:

1. Registrace přes devise, facebook, google, twitter
2. Editace profilu
    - jméno, příjmení, ulice + číslo popisné, PSČ, Město, Email, Heslo
3. Zadáni/editace předmětu (pronajímatel)
    - Název, Typ (movitá věc, nemovitost, ostatní), Popis, 0-N fotografií
    - Označení, zda je věc veřejná/vyhledatelná nebo ne
    - Označení, zda v případe, že je předmet bez aktuální smlouvy se dá nabídnout v katalogu na pronájem
4. Zadáni/Ukončení smlouvy (pronajímatel)
    - datum od, datum do
    - nepovinný popis
    - druhá strana - zadá email, pokud existuje user na druhé straně (nájemce), tak se spáruje
    - pokud neexistuje, nabídne mu to vyplnění dalších věcí (jméno, adresa, telefon...) s bold infem, že se jedná o údaje o 3. osobě a je povinností pronajímatele zajistit souhlas se zpracováním těchto dat
    - označení, zda smlouva je veřejná nebo ne
    - editace + ukončeni, pozor ukončením se nemaže, ale jen označuje že je ukončena, aby zůstala historie
5. Vyhledáváni - veřejná funkce - pouze ve veřejných předmětech/smlouvách
    - ve výsledcích vyhledávání přidat kontaktní formulář na vlastníka - nahlášení problémů
6. Hlídací pes - upozornění na výsledky hledáni v denních/týdenních intervalech + sledování změn určitého předmětu

Takhle vypadalo základní zadání funkcí, které se ale časem rozšiřovalo. Začala jsem tím, že jsem založila novou Rails aplikaci a zprovoznila přihlašování prvně jen přes gem Devise a přihlašování přes facebook, gmail a twitter jsem nechala až ke konci. Dále jsem udělala zatím alespoň základní design pro lepší přehlednost projektu.

Poté sem zprovoznila přidávání předmětů a vkládání obrázků k předmětům z počítače, nebo formou zadání adresy URL. Po domluvě s konzultantem jsem nastavila vkládání na maximum jen 6 obrázků.

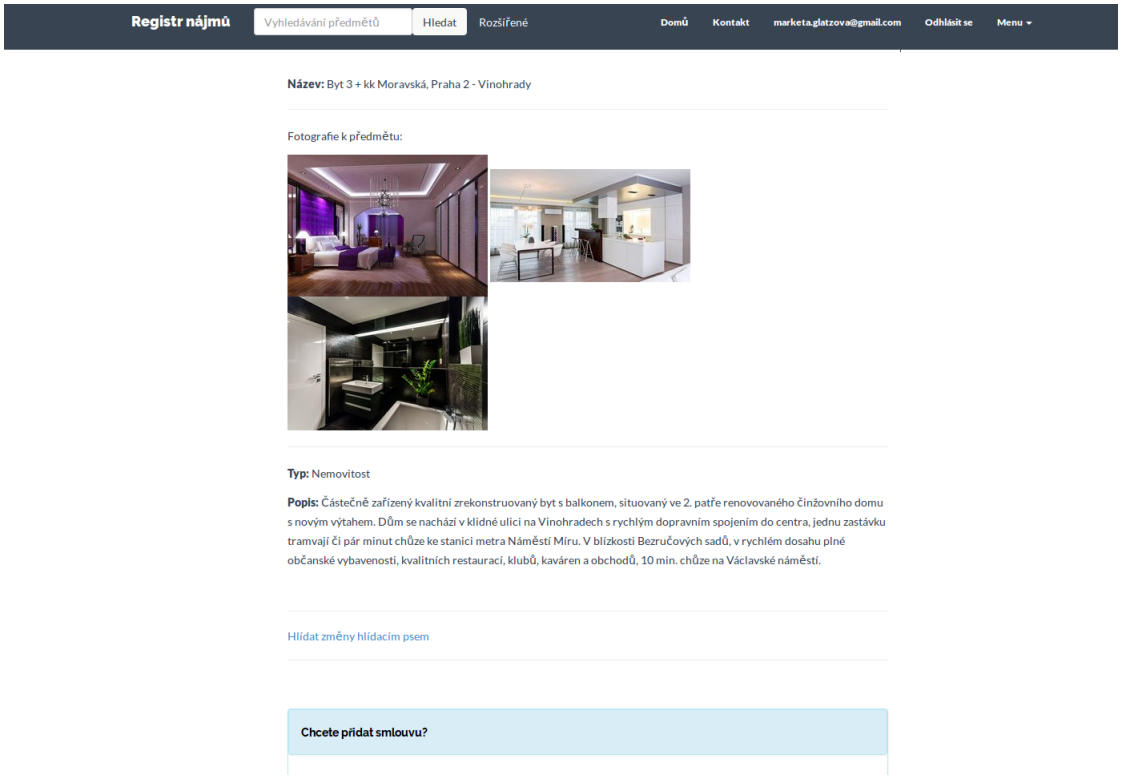
U každého předmětu ve view vidíme jestli je na předmětu smlouva veřejná, nebo neveřejná a majitel předmětu může vkládat smlouvy k jeho předmětům. O smlouvách Vám ještě níže povím více.

Pokud nejsou na předmětu žádné smlouvy může vlastník ve view kliknout na: "Nabídnout předmět k pronájmu"a bude se zobrazovat v katalogu pro pronájem, ale také nám vyskočí ještě informace, že předmět musí být nastavený jako veřejný aby se v katalogu zobrazoval. Zobrazení předmětu v katalogu pro pronájem lze kdykoliv zrušit a je to řešeno tak, že po kliknutí na tento odkaz se proměnná typu boolean nastavuje na true nebo false.

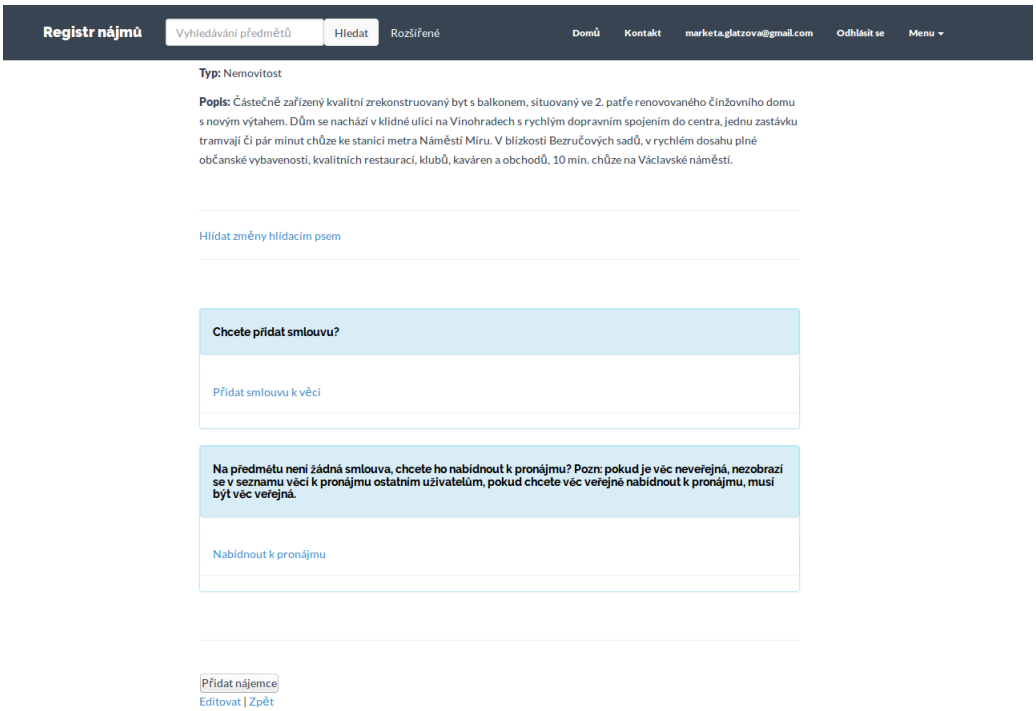
Každý přihlášený uživatel má k dispozici stránku všech volných předmětů k pronájmu a také předmětů, které si pronájímá on sám. Ve view u předmětu lze také přidat nájemce, nebo odebrat nájemce. Po kliknutí na tlačítko "Přidat nájemce"nás to odkáže na novou stránku, kde je vyžádána emailová adresa nájemce a nastávají dvě varianty. Pokud nájemce již v databázi je tak se pomocí id ihned prováže s předmětem jako osoba, která si tento předmět pronajímá. Pokud však nájemce ještě v databázi není, vyskočí vlastníkovi předmětu formuláře pro zadání základních informací o nájemci, ale pod formulářem musí zaškrtnout prohlášení, že zajistil souhlas se zpracováním těchto dat.

Po kliknutí na odeslat se již nájemce s předmětem spáruje. Odebrání nájemce probíhá již velmi jednoduše stačí pouhé kliknutí na tlačítko: "Odebrat nájemce". Jméno nájemce je neveřejná informace a zobrazuje se pouze danému nájemci, nebo vlastníkovi daného předmětu, ostatní uživatelé vidí pouze oznámení, že je u předmětu již nějaký nájemce.

Další "vychytávkou"u předmětu je pro registrované uživatele možnost kliknutí na tlačítka: "Hlídat přemět hlídacím psem". Zobrazí se nám upozornění, že hlídací pes byl úspěšně zapnut a od této doby nám budou chodit na naší emailovou adresu zadanou při registraci veškeré upozornění o změnách na předmětu. Hlídací pes lze kdykoliv zrušit.

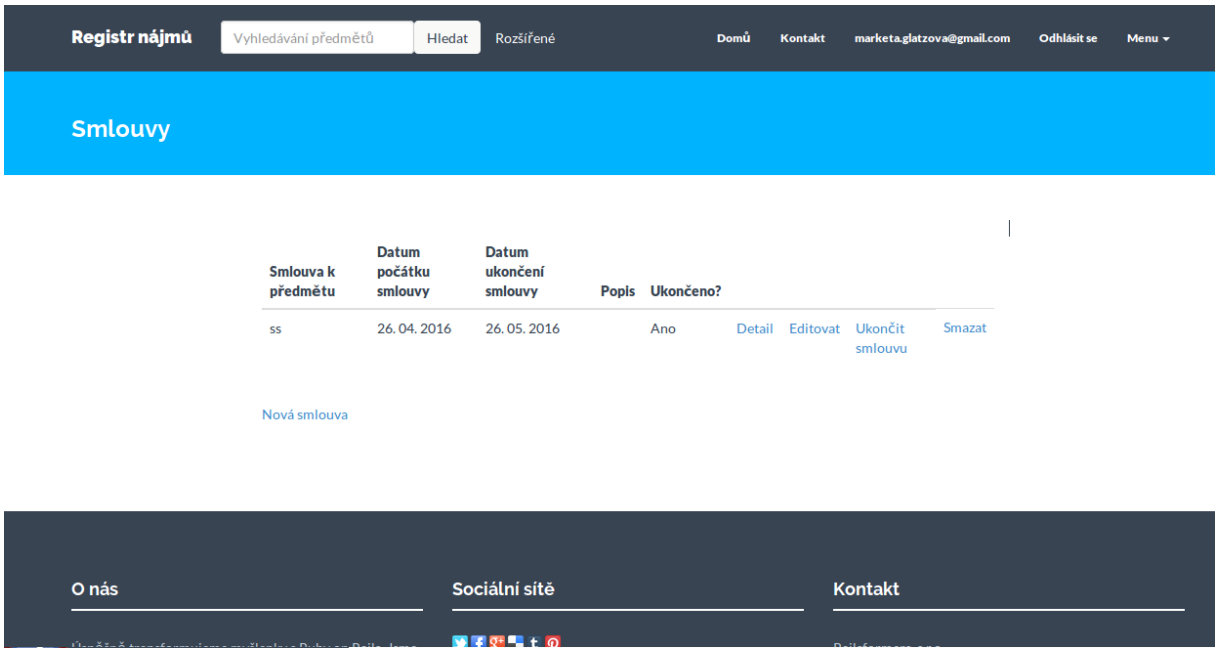


Obrázek 6: Registr nájmů, stránka předmětu



Obrázek 7: Registr nájmů, stránka předmětu 2

Nyní bych se ráda vrátila k přidávání smluv, které jsem pomocí id provázala s přeměty. Při vkládání nové smlouvy vybereme předmět ke kterému patří, zadáme datum od kdy, do kdy je smlouva v platnosti a můžeme ještě přidat nepovinný popis smlouvy. Datum začátku smlouvy musí být menší, než datum ukončení smlouvy jinak nám to vyhodí chybu, že musíme datum upravit.

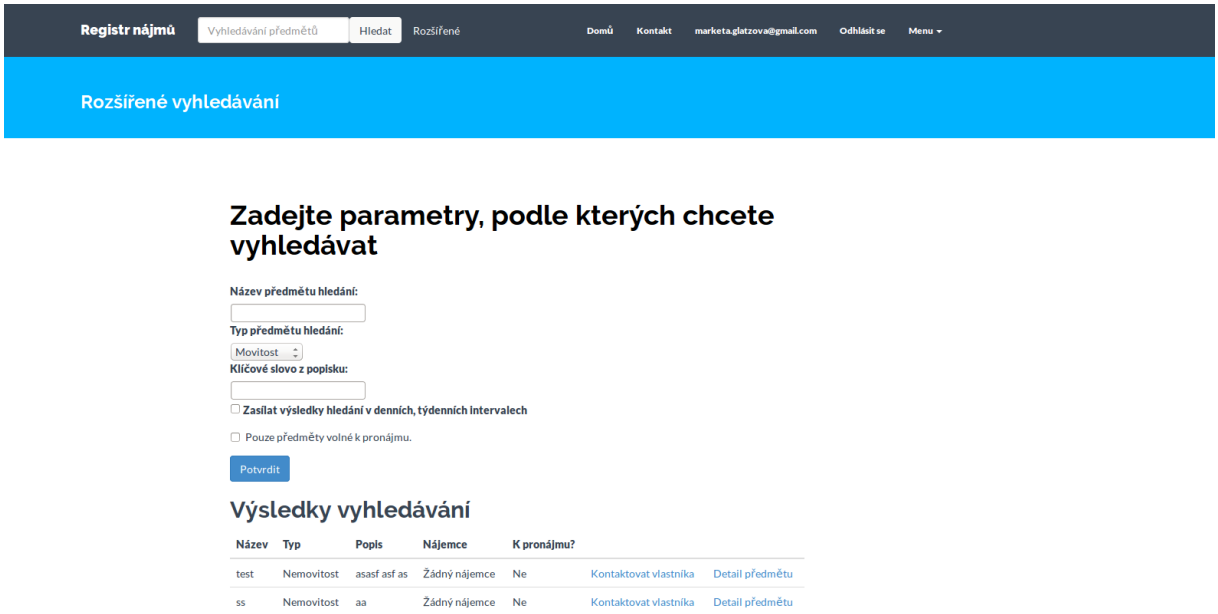


Obrázek 8: Registr nájmů, smlouvy

Ve view ve kterém zadáváme novou smlouvu nebo předmět najdeme také checkbox pro zaškrnutí zda je věc, nebo smlouva veřejná. Přidala jsem tedy do databáze další proměnnou typu boolean s názvem is_public, která je defaultně nastavená na false a pokud je checkbox zaškrtnutý, nastaví se hodnota na true a pokud není zůstává false. Pokud máme například veřejný předmět a neveřejnou smlouvu tak se nám ukáže u předmětu upozornění, že je na něm zadaná neveřejná smlouva. Podobně jsem také řešila ukončování smlouvy akorát s tím rozdílem, že sem použila další boolean proměnnou s názvem terminate ale ukončování probíhá přímo ve výpisu smluv po kliknutí na: "Ukončit smlouvu".

Ukončená smlouva se již dále nezobrazuje u předmětu, ale zůstává dále v evidenci jako ukončená a zadavatel ji stále vidí v jeho seznamu smluv, ale již jako ukončenou.

Všechny předměty lze vyhledávat dvěma způsoby stejně jako u projektu hodnocení advokátů a je to vlastně jediná funkce, kterou může provádět neregistrovaný uživatel. Vyhledávání probíhá pouze ve veřejných předmětech a předměty neveřejné vidí pouze vlastník těchto předmětů. U rozšířeného vyhledávání můžeme hledat podle názvu předmětu, typu předmětu(movitost, nemovitost, ostatní), klíčového slova z popisku a nebo lze také vyhledávat jen předměty volné k pronájmu.



Obrázek 9: Registr nájmů, vyhledávání

Před potvrzením hledání mohou ještě registrovaní uživatelé zaškrtnout check box: "Zasílat výsledky hledání v denních, týdenních intervalech"a budou dostávat pravidelně výsledky vyhledávání na svou emailovou adresu zadanou při registraci. Zaškrtnutím checkboxu se do tabáze uloží všechny vyplněné položky a pravidelně se projíždí databáze a výsledky se zasílají emailem.

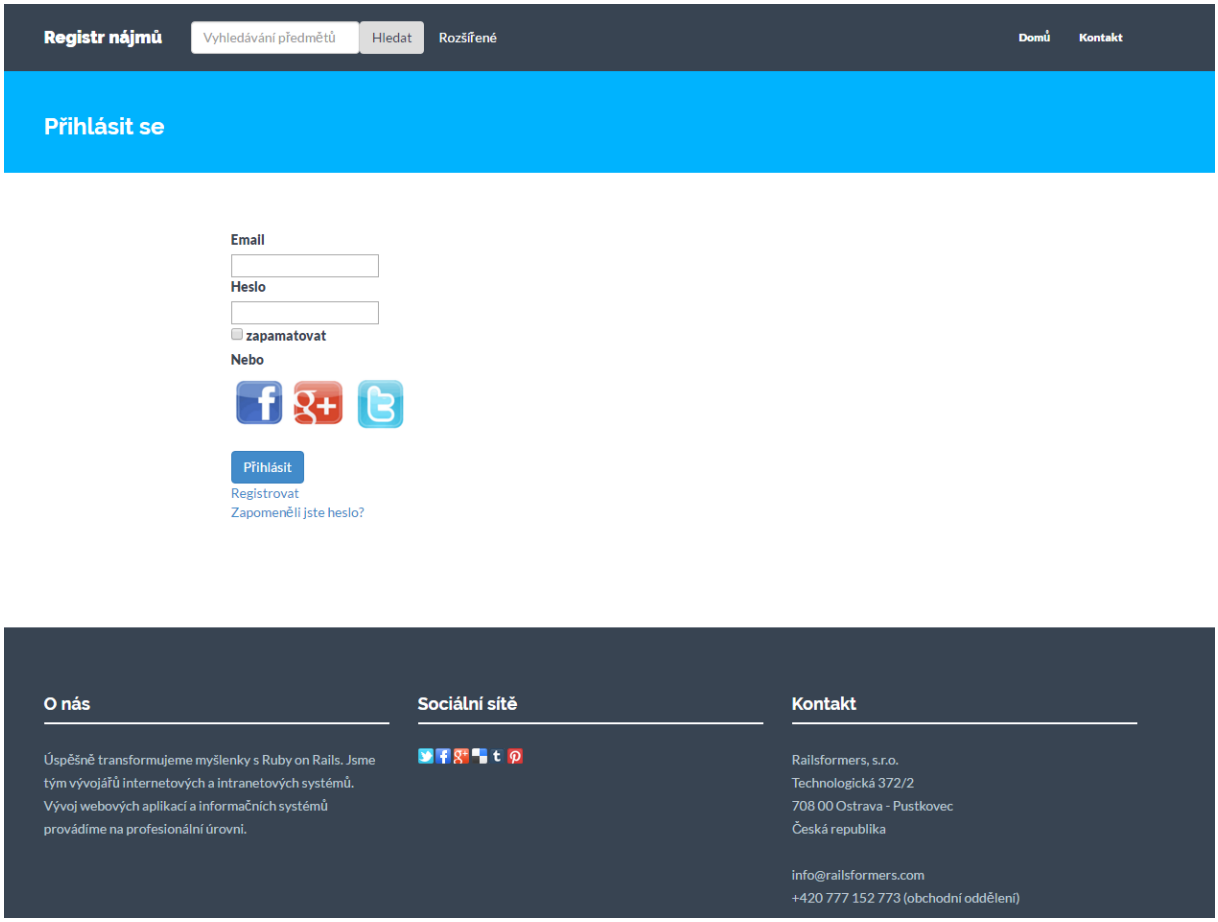
Pod formulářem vyhledávání vidíme výsledky vyhledávání ve kterých můžeme u každého předmětu kliknout také na: "Kontaktovat vlastníka"a odkáže nás to na další stránku, na které již stačí zadat pouze předmět a obsah zprávy. Email příjemce a odesílatele není třeba nikde zadávat jelikož kontaktovat vlastníka může jen uživatel, který se zaregistroval a při registraci již email zadal a vlastník předmětu je také již registrovaná osoba. Neregistrovanou osobu kliknutí na tento odkaz odkáže k přihlašovacímu formuláři.

Každý uživatel má také k dispozici stránku "Moje zprávy", kde vidí přehled všech svých odeslaných a přijatých zpráv pro lepší přehled aby nemusel vše dlouze hledat ve své emailové schránce.

K zasílání všech emailů nám pomáhá gem 'mail_form'. Pomocí tohoto gemu zasílám také uživateli po registraci na stránkách uvítací email s přihlašovacími informacemi.

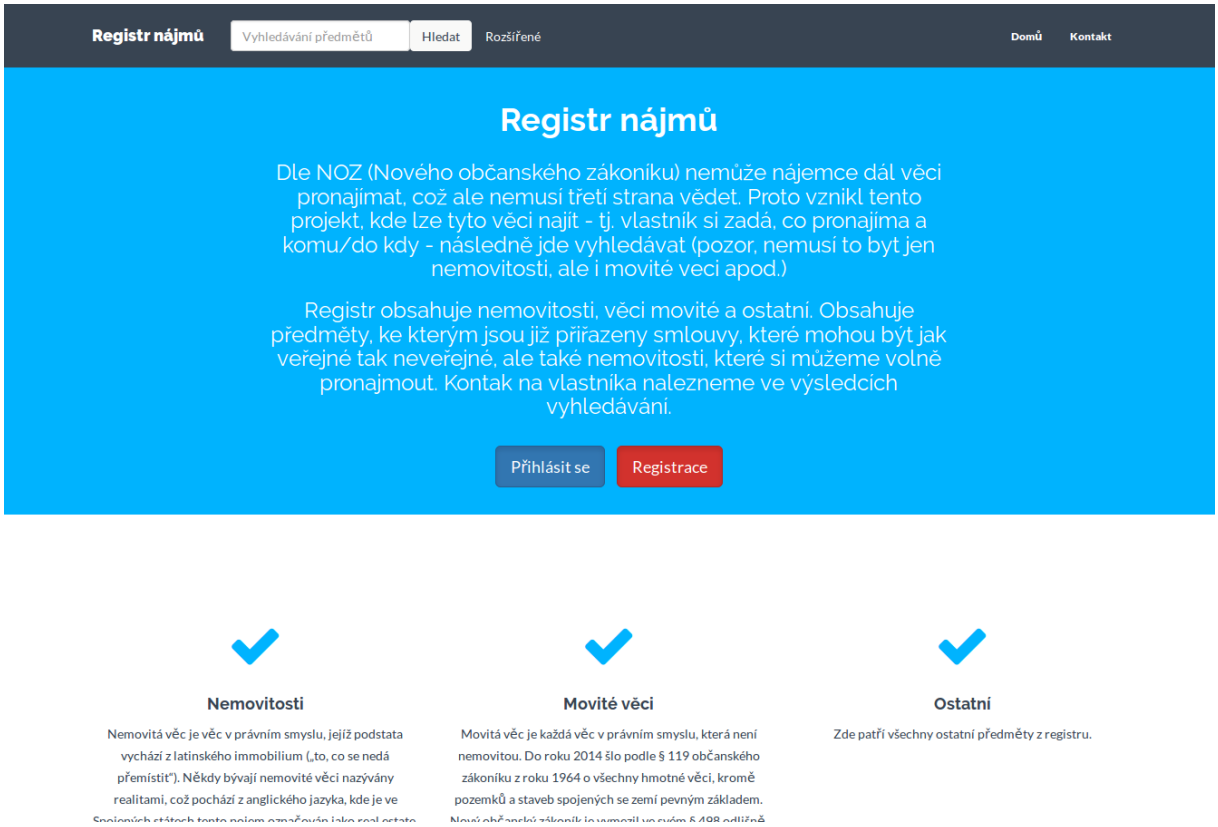
Každý uživatel má také stránku se svým profilem, který se vytvoří automaticky při registraci a hned po registraci je uživatel odkázán k jeho vyplnění, ale není to povinné.

Ke konci jsem se pustila do přihlašování pomocí facebooku, gmailu a twitteru k čemuž mi pomohly gemy 'omniauth-facebook', gem "omniauth-google-oauth2"a gem "omniauth-google-twitter". Přes facebook a gmail se lze také i přihlašovat a ne pouze registrovat. Aby přihlašování a registrace správně fungovaly musela jsem si na strankách facebooku, googlu a twitteru aplikaci zaregistrovat vygenerovat potřebné údaje.



Obrázek 10: Registr nájmů, přihlašování

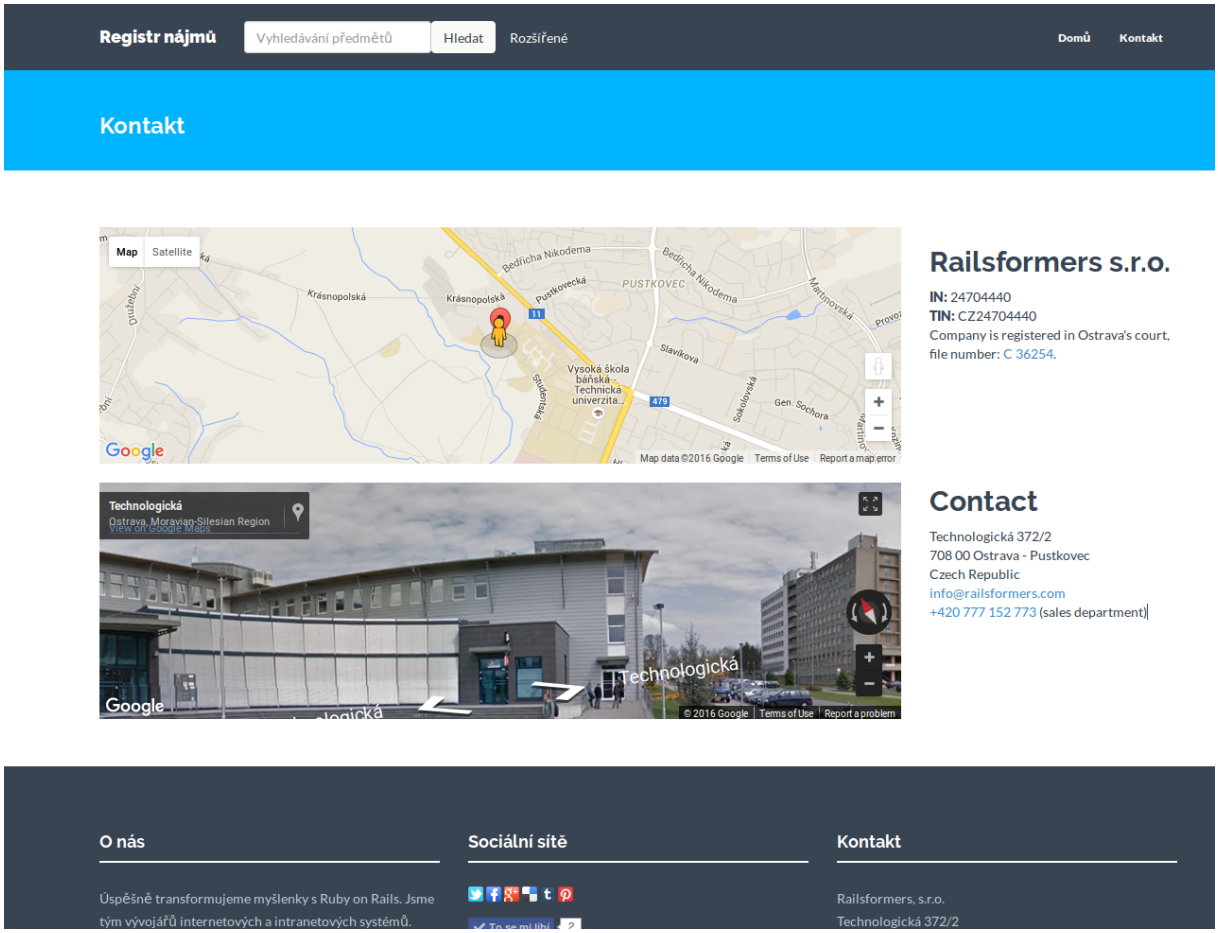
U projektu registr nájmu jsem se také rozhodla udělat pěkný vzhled výsledných WWW stránek a proto jsem používala bootstrap abych na internetu nalezla více různých šablon(templates), které jsou zdarma a pomocí kterých jsem dosáhla již uspokojivého designu.



Obrázek 11: Registr nájmů, úvod

Vybranou šablonu jsem dále upravovala podle svých požadavků a vytvářela také nové třídy v CSS souborech, které mi pomohly toho docílit. Poté jsem ještě na stránky přidala také sociální tlačítka pro like a sdílení, které by podle mě měly být součástí všech moderních webovek.

Na úvodní stranu jsem přidala informace o projektu, ale také informace o firmě, dále na ní můžeme vidět seznam pěti posledních přidaných předmětů a podobně. Přidala jsem také klasickou stránku s kontaktem, která obsahuje i mapy řešené pomocí javascriptu a odkaz na tuto stránku se nachází vedle odkazu na domovskou stránku.



Obrázek 12: Registr nájmů, kontaktní strana

Úplně posledním krokem byly testy a překlady, kterým se věnuji již v samostatných kapitolách z důvodu rozsahu.

Při tvorbě tohoto projektu jsem narazila na různé problémy, ale nejvíce času mi vzalo to, že jsem původně špatně pochopila zadání a bylo mi vysvětleno až po sléze. V zadání jsem mněla zadáno, že máme dvě role a proto tedy v původní verzi projektu jsem rozdělovala uživatele na dvě různé skupiny a to na role nájemníka a pronajímatele. Při registraci si uživatel vybral svou roli, ale první problém nastal při registraci přes facebook. Tento problém jsem vyřešila tak, že jsem nastavila při této formě přihlašování defaultní roli, která se následně dala změnit na stránce, kde uživatel má možnost měnit také heslo a pod. Později mi však bylo vysvětleno, že nájemce může být také zároveň pronajímatel a rozdělení na role jsem musela z projektu úplně odstranit.

3.5 Překlady

Ruby i18n (zkratka pro internacionalizaci) je gem, který je dodáván s Ruby on Rails (počínaje verzí Rails 2.2). Poskytuje snadno použitelný a rozšiřitelný rámec pro překládání naši aplikace do jednoho jiného jazyka, než je angličtina, nebo pro poskytování multi-jazykové podpory aplikace.

Internacionalizace je složitý problém. Přirozené jazyky se liší v mnoha ohledech (např pravidlech pluralizace), proto je velice obtížné poskytnout nástroje pro řešení všech problémů najednou. Z tohoto důvodu se Rails i18n API zaměřuje na:

- poskytování podpory pro angličtinu a podobné jazyky
- usnadňuje přizpůsobování a rozšiřování všeho pro ostatní jazyky

V rámci tohoto řešení, každý statický řetězec v Rails jako například Acive Record validační zpráva, čas a formát data byl internacializován, takže lokalizace v Rails aplikaci znamená v první řadě tyto výchozí hodnoty.

To znamená, že nejdůležitějšími metodami v i18n API jsou:

- překlady => Vyhledávání textovách překladů
- lokalizace => Lokalizuje datumy a časové objekty do lokálních formátů

Slouží nám k tomu dva aliasy, které můžeme používat například následovně:

```
I18n.t 'subject.title'
I18n.l Time.now
```

Výpis 16: Aliasy pro překlady a lokalizace

Ruby i18n gem je také rozdělen do dvou částí:

- Veřejné API rámce i18n - modul Ruby s veřejnými metodami, které definují, jak knihovna funguje
- Výchozí backend (který je záměrně pojmenovaný Simple backend), který implementuje tyto metody

Jako uživatel byste měli mít vždy přístup pouze do veřejné metody i18n modulu, ale je užitečné vědět o možnostech backendu.

Prvním důležitým krokem pro tvorbu překladů je vložit do projektu gem, který nám překlady umožní.

```
gem 'rails-i18n'
```

Výpis 17: Gem pro překlad

Po instalací gemu, spuštěním bundle install můžeme provádět různé konfigurace. Například jsem do složky config do souboru application.rb vložila následnující řádek kódu:

```
config.i18n.default\_locale = :cs
```

Výpis 18: Nastavení výchozího jazyka

Tato část kódu nám nastavuje výchozí jazyk na češtinu. Ve svých projektech jsem nerealizovala multi-jazykovou podporu, ale překládala jsem stránky vždy do češtiny.

Soubory s překlady se zpravila nacházejí ve složce config/locales a mají koncovku .yml například: cs.yml a en.yml jsem používala nejčastěji. Dále zde například můžeme mít devise.cs.yml kde můžeme nastavit překlady do češtiny pro gem Devise a pod.

Příklad překladu může vypadat například takto:

```
cs:
hello: "Ahoj!"
```

Výpis 19: Způsob zapisování překladů v souboru typu .yml

Tento překlad dále použijeme v národním prostředí(locales) následovně pomocí použití 'I18n.t':

```
I18n.t 'hello'
```

Výpis 20: Národní prostředí

Ve views postačí použití prostého 't':

```
<%= t('hello') %>
```

Výpis 21: Views překlady

3.6 Testování

Každá správná aplikace by měla projít testováním, než se uvede do reálného provozu aby se předešlo zbytečným chybám, které může člověk lehce přehlédnout.

Pro tvorbu testů jsem používala především gemy rspec-rails, capybara a factory_girl_rails. Pomáhala mi převážně kniha, kterou jsem četla již na začátku, ale studovala jsem plno věcí, které v knize chyběly na internetu.

Do Gemfilu sem přidala následující gemy.

```
group :development, :test do
  gem 'byebug'
  gem "rspec-rails", "~> 3.1.0"
  gem "factory_girl_rails", "~> 4.4.1"
end
group :test do
  gem "faker", "~> 1.4.3"
  gem "capybara", "~> 2.4.3"
  gem "database_cleaner", "~> 1.3.0"
  gem "launchy", "~> 2.4.2"
  gem "selenium-webdriver", "~> 2.43.0"
end
```

Výpis 22: Gemy pro testování

Instalace ve vou skupinách je proto, že rspec-rails a factory_girl_rails se používají jak v rozvoji tak v testovacím prostředí. Zbývající drahokamy jsou používány pouze tehdy, pokud skutečně spouštíme naše testy takže není nutné je načítat i ve vývoji. To nám také zaručuje, že drahokamy používané výhradně pro generování kódu, nebo spouštění testů nejsou nainstalovány v provozním prostředí při nasazování na server.

- byebug => lze zavolat kdekoliv v kódu pro zastavení operace a získání debuggeru v konzoli
- rspec-rails => nám přidá do naší aplikace specifické funkce, které potřebujeme pro testování aplikace
- factory_girl_rails => nahrazuje výchozí příslušenství Rails pro dodávání testovacích dat do testovací sady pro lepší testování(podpora pro sestavení více strategií, podpora pro více továren stejné třídy např.: role)
- faker => generuje jména, e-mailové adresy a další
- capybara => pomáhá simulovat interakce uživatelů a můžeme pomocí ní procházet web a zkoušet různé přístupy na jednotlivé stránky a podobně
- database_cleaner => pomáhá nám zajistit aby byla databáze před spuštěním testů čistá
- launchy => tento gem nám zvládá otevřít výchozí webový prohlížeč a na požádání nám ukázat co zrovna vykresluje, což může být užitečné při ladění
- selenium-webdriver => potřebujeme, pokud chceme testovat části napsané v JS, interakce s capybarou

Po úspěšném nainstalování těchto gemů spuštěním bundle install v konzoli se můžeme pustit do nastavení testovací databáze.

Databázi nalezneme v config/database.yml

```
#Nastaveni pro SQLite3

test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

#Nastaveni pro MySQL

test:
  adapter: mysql2
  encoding: utf8
  reconnect: false
  database: contacts_test
  pool: 5
  username: root
  password:
  socket: /tmp/mysql.sock
```

Výpis 23: Natavení databáze pro testování aplikace

Databázi ještě musíme vytvořit spuštěním příkazu rake db:create:all a můžeme již vygenerovat potřebné složky a soubory pro naše testy spuštěním příkazu rails generate rspec:install

Generátor nám ukáže následující informace :

- create .rspec
- create spec
- create spec/spec_helper.rb
- create spec/rails_helper.rb

Vidíme tedy, že se nám vygenerovaly i pomocné sobory.

Po tomto kroku stačí už jen do config/application.rb přidat následující nastavení:

```
config.generators do |g|
  g.test_framework :rspec,
  fixtures: true,
  view_specs: false,
  helper_specs: false,
  routing_specs: false,
  controller_specs: true,
  request_specs: false
  g.fixture_replacement :factory_girl, dir: "spec/factories"
end
```

Výpis 24: Nastavení generátorů pro testy

Tímto se konečně dostáváme do fáze, kdy můžeme pomocí příkazu rspec spec v konzoli spustit testy. Později lze spouštět třeba jen testy u modelů pomocí rspec spec/models. V této fázi máme zatím nula příkladů a nula selhání, přejdu tedy k základním testům.

Testy se vkládají do podsložek složky spec. Testy pro controllery se vkládají do složky controllers, pro modely do models, pro mailery do mailers, capybara testy do features a soubory používající FactoryGirl do factories. Můžeme mít i více složek, ale pro základní testy myslím, že jsou tyhle dostačující.

Na praxi jsem testovala hlavně modely a přístupy na stránky pomocí capybary. V následujících podkapitolách vám předvedu pár ukázek testů.

3.6.1 FactoryGirl

```
require 'faker'

FactoryGirl.define do
   sequence :email do |n|
     "email#{n}@factory.com"
   end
 factory :user do
   email { Faker::Internet.safe_email }
   password '12345678'
 end
end
```

Výpis 25: Ukázka FactoryGirl

V této ukázce je pro vytváření usera využit gem faker, který generuje emaily a heslo je nastaveno jednoduše na '12345678'. Tento kousek kódu stačí k vytvoření testovacího modelu, se kterým dále pracujeme třeba v testech u modelu.

3.6.2 Testování modelů

Testy u modelů předvedu na následujících kódech.

```
require 'spec_helper'

RSpec.describe User, type: :model do

before(:each) do
  @user = FactoryGirl.create(:user)
end

describe "relation" do
  it {should have_one :profile}
end
describe "validation" do
  it {should validate_presence_of :email}
  it {should validate_presence_of :password}
end
```

Výpis 26: Ukázka testování modelu

Na této ukázce vidíme, že před spuštěním testů se zavolá FactoryGirl pro vytvoření testovacího subjektu.

Dále zde můžeme vidět část "relation"ve které se nastavují vztahy, v tomto případě vidíme, že uživatel má jeden profil.

V části "validation"můžeme pozorovat jak se tvoří jednoduše pomocí validate_presence_of testy pro kontrolu přítomností parametrů, které máme nastavené v daném modelu. Pro projití testů je tedy nutné aby byly obsaženy parametry email a password(heslo).

Na další ukázce předvedu další testy.

```
describe "invalid when attributes are empty" do
  it "is invalid when email is empty" do
    @user.email = nil
    expect(@user).to_not be_valid
  end
  it "is invalid when password is empty" do
    @user.password = nil
    expect(@user).to_not be_valid
  end
end
  describe "is invalid when password has less than 4 character" do
    it { should ensure_length_of(:password).is_at_least(4) }
  end
```

Výpis 27: Ukázka testování modelu 2

V této další ukázce testujeme jestli není atribut prázdný. Jestli se rovná nil tak není validní(zajištěno pomocí to_not be_valid).

Poslední část ukazuje jakým způsobem se dá v modelu otestovat například počet znaků hesla. Počet znaků zjístíme pomocí should ensure_length_of, v závorce máme název parametru s dvojtečkou a voláme ještě metodu is_at_least, která zjístí minimalní počet znaků uvedený v závorce. Tento test projde, pokud bude mít heslo alespoň 4 znaky.

3.6.3 Capybara

Nakonec bych ještě ráda ukázala pár testů pomocí Capybary.

```
let!(:user) { create(:user) }
let!(:subject_not_public) { create(:subject, is_public2: false) }
let!(:my_subject) { create(:subject, is_public2: false, user_id: user.id) }
let!(:subject_is_public) { create(:subject, is_public2: true) }
let!(:my_subject2) { create(:subject, is_public2: true, user_id: user.id) }
```

Výpis 28: Ukázka testování capybara

Zde vidíme jak se vytváří jednotné případy, které mohou nastat například u předmětů v projektu registr nájmů a které potřebujeme otestovat. Tyto případy musíme sami vymyslet a napsat, pomáhá nám při tom let!.

Existuje let a let! Let používáme, když definujeme časovou pomocnou metodu. Hodnota bude uložena do mezipaměti napříč voláními ve stejném příkladu, ale ne přes příklady. Let je později vyhodnocený tzn. není hodnocen dokud není poprvé volána nadefinovaná metoda. Let! můžeme použít k vynucení volání metody před každým příkladem.

```
describe 'visiting subjects when signed out and subject is not public' do
 it 'should get not public message' do
   visit  subject_path(subject_not_public)
   should have_content(I18n.t('not_public'))
 end
end
```

Výpis 29: Ukázka testování capybara

Následně poté voláme vytvořený případ a navštívíme stránku s tímto případem pomocí visit a poté pomocí should have_content jen ověříme, jestli se nám zobrazil na této stránce správný výpis.

Takhle bych zjednodušeně vysvětlila jak funguje capybara. Capybara má ale spoustu dalších funkcí, dokáže nejen navštěvovat stránky, ale také klikat na tlačítka, provádět testovací registrace a podobně. Je to velice užitečný gem při tvorbě webů.

4 Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné v průběhu odborné praxe

Znalosti a dovednosti, které jsem na praxi uplatnila nebylo mnoho. Většina věcí pro mne byla úplnou novinkou.

Využila jsem ale například mé základní znalosti JavaScriptu které jsem získala z předmětu Tvorba mobilních aplikací I(TAMZ1). Z tohoto přemětu jsem nevyužila jen základy JS, ale také základy HTML.

Dále jsem využila znalosti z přemětu Uživatelská rozhraní(URO) ve kterém jsem se naučila jak by mělo správně GUI vypadat. A to ne jen aby bylo přehledné, ale také jsem se v tomto předmětu naučila, které kombinace barev používat a podobně.

Poslední uplatněné znalosti se týkaly databází, se kterými jsem se alespoň letmo v pár předmětech setkala a znala jsem tedy alespoň základy fungování databází.

5 Znalosti či dovednosti scházející v průběhu odborné praxe

Znalosti a dovednosti, které mi v průběhu praxe scházely bylo opravdu dost. Předtím jsem Ruby a Ruby on Rails neznala a také jsem neměla žádné zkušenosti s vývojem webových aplikací. Jelikož jsem předtím tedy žádné webové aplikace nikdy nevyvíjela tak s tím souvisely také další chybějící znalosti a to například znalosti CSS a HAML. Testování a tvorba překladů pro mne bylo také novinkou stejně jako práce s Gitem.

6 Závěr

Jsem velmi ráda za možnost vykonání odborné praxe ve firmě Railsformers. Díky této praxi jsem nabyla spoustu nových znalostí, také jsem si vyzkoušela jak to vlastně ve firmě funguje a zkusila si pracovat i v týmu. Tyto znalosti jistě ještě dále využiju.

Markéta Glatzová

Literatura

[1] Railsformers, s. r. o., *D*ostupné na: http://railsformers.com/.

[2] Projekt Hedurio, *D*ostupné na: https://hedurio.com/.

[3] Projekt sMoneyBox, *D*ostupné na: http://www.smoneybox.com/cs.

[4] Projekt sÚčto, *D*ostupné na: https://www.sucto.cz/.

[5] Projekt sRecepty, *D*ostupné na: http://www.srecepty.cz/.

[6] Projekt Projektově, *D*ostupné na: https://www.projektove.cz/.

[7] Git, *D*ostupné na: https://git-scm.com/book/cs/v1/%C3%9Avod-Z%C3%A1klady-syst%C3%A9mu-Git.

[8] Guide, *D*ostupné na: http://guides.rubyonrails.org/

[9] Railscasts videa, *D*ostupné na: http://railscasts.com/

[10] jQuery Raty, *D*ostupné na: https://github.com/wbotelhos/raty

[11] Návod na blog, *D*ostupné na: http://guides.rubyonrails.org/getting_started.html

[12] Aaron Sumner, *E*veryday Rails Testing with RSpec : Návod jak psát testy v RSpec

[13] Mark G. Sobell, *M*istrovství v Linuxu, příkazový řádek, shell, programování

[14] Steven Holzner, *Z*ačínáme programovat v Ruby on Rails

[15] Sam Ruby, *R*uby on Rails, průvodce agilním vývojem webových aplikací